On Evolving Partitioned Web Service Orchestrations

Fdhila Walid, Rinderle-Ma Stefanie, Aymen Baouab, Olivier Perrin, Claude Godart

On Evolving Partitioned Web Service Orchestrations

Walid Fdhila, Stefanie Rinderle-Ma
University of Vienna, Austria,
Faculty of Computer Science
{walid.fdhila, stefanie.rinderle-ma}@univie.ac.at

Aymen Baouab, Olivier Perrin, Claude Godart
University of Lorraine,
Nancy, France
{aymen.baouab, olivier.perrin, claude.godart}@loria.fr

***Abstract*—Many researches argue that centralized Web Service (WS) orchestrations stop short in dealing with key requirements such as scalability, privacy and reliability. Consequently, fragmentation and decentralization have been proposed to overcome these limitations. In detail, the centralized orchestration is fragmented into behaviorally equivalent distributed partitions such that their combined execution recreates the function of the original orchestration. However, the evolving nature of business processes created the need for an efficient change support. Since the decentralization leads to the distribution of the activities, the control and data flows, it becomes difficult to specify the changes directly on the derived partitions. Therefore, it is more judicious to specify the changes on the centralized orchestration model and propagate them to the derived partitions. In this paper, we propose a comprehensive change framework for partitioned WS orchestration scenarios and demonstrate how to specify and propagate the changes from the centralized model to its resulting decentralized partitions.**

***Keywords*-decentralization, business process, change propagation, web service.**

I. Introduction

Globalization and the increase of competitive pressures created the need for agility in business processes, including the ability to outsource, offshore, or otherwise distribute its once-centralized business processes or parts thereof [1]. In this sense, many works were proposed to partition a composite web service [2], [1]. The partitioning transforms the centralized process into behaviorally equivalent distributed partitions such that their combined execution recreates the function of the original orchestration. The flexibility introduced by the decentralization on the other hand raises necessary requirements like adaptation to change. Changes may range from simple modifications to a complete restructuring of the business process to improve efficiency. In the context of the decentralized service orchestrations, applying these changes in a straightforward manner on the derived orchestration partitions is a complex maintenance task, since the control and data flows are decomposed over multiple partitions. Moreover, changing a derived partition may affect the way it interacts with others. In this sense, we have been investigating change propagation in decentralized composite web services [3]. Given a well-behaved structural update on a centralized orchestration, our approach *automates the change forward propagation* that consistently propagates the update to the derived decentralized partitions. The main advantage of this method, is that only partitions concerned by the change are affected, and there is no need to recompute the whole decentralization or redeploy all the partitions.

The work presented in this paper has been partly conducted within the project I743 funded by the Austrian Science Fund (FWF).

This paper is an extended and revised version of our previous work [3]. With respect to this version, the extensions include the adopted change patterns, a revision of the actions for change propagation, the theoretical evaluation of the presented method and some implementation details.

The remainder of this paper is structured as follows. Section II illustrates and motivates the importance of change propagation in decentralized orchestrations. While Section III presents the formal definitions, Section IV details the change propagation mechanism. Section V evaluates the approach and Section VI presents the related works. Finally, Section VII summarizes the contribution and outlines future directions.

II. Motivation

To motivate and illustrate the methods presented in this paper, we make use of the sample orchestration (cf. Figure 1) presented in [4]. This orchestration model encodes a claims handling process at an insurance company IC. For more details about the example, the readers may refer to [4].

This centralized model presents many drawbacks since all interactions between the services are channeled through IC. The partitioning consists in splitting the latter into small partitions each of which executed by a separate orchestrator. The process is split according to a criterion, such as each partition include only activities which have the same properties (e.g. privacy, role, optimization, etc.). For instance, one could assign critical activities to the same partition to be executed by a high secure orchestrator while assigning others to a less secure ones. Figure 2 depicts a possible decentralized execution settings for IC which is split into three partitions $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$. Each partition includes a subset of the initial activities and extra activities to communicate with other partitions (interaction activities). The connectivity between activities of the centralized process is translated to that between activities of different partitions, through message exchanges. Yellow activities represent data exchanges and gray activities are control flow connections. The partitioning of the centralized process model example uses the techniques presented in [2].

Now, lets consider the process model in Figure 1 and assume that IC wants to replace the fragment $\mathcal{F}$ by the new fragment $\mathcal{F}\prime$. In this change, the choice patterns (g_4 and g_5) are replaced

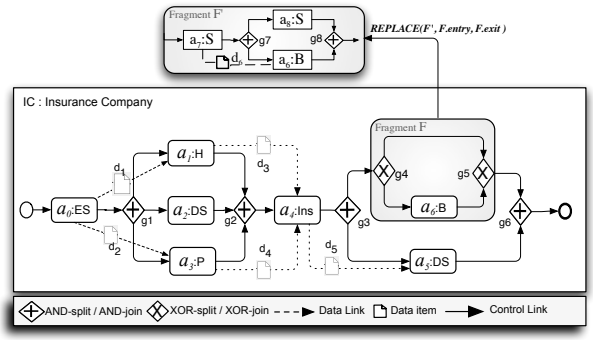

Fig. 1. Insurance Claims Handling Process Example

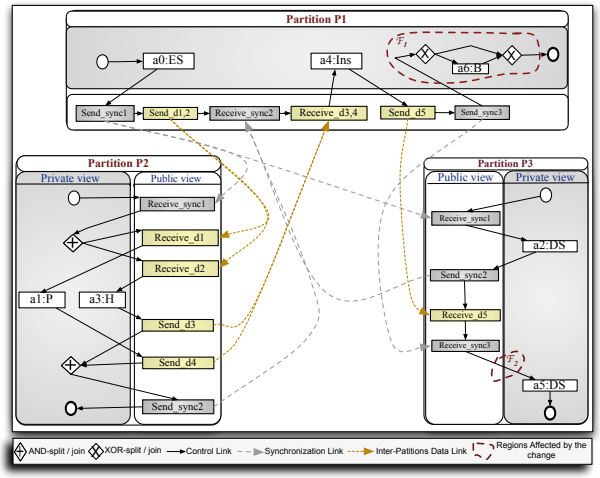

Fig. 2. Decentralized Insurance Claims Handling Process

by parallel patterns (g_7 and g_8) and two new activities a_8 and a_9 invoking the same service S are added as well as a data dependency between a_8 and a_6. We assume that the added activities have the same properties as the activities of partition P_1 and therefore would be assigned to $\mathcal{P}_1$. The activity $a_6 : B$ is also concerned by the change since it becomes in parallel with a_9 and consequently the replacement would mainly affect the partitions $\mathcal{P}_1$ and $\mathcal{P}_3$ (c.f. Figure 2). In Figure 2, the regions $\mathcal{F}_1$ and $\mathcal{F}_2$ in partitions $\mathcal{P}_1$ and $\mathcal{P}_3$ respectively, represent the fragments which are affected by the change. It should be noticed that handling the changes locally and propagating them avoid re-decentralizing the whole process. Indeed, re-decentralizing the process model leads to the re-deployment of all partitions (even those not affected by the change). In this process example, the partition $\mathcal{P}_2$ is not concerned by the change and would not be re-deployed.

III. Preliminaries

In order to provide a generic approach for change propagation in partitioned composite services, we adopt a high level reasoning using an abstract notation. A process model specifies the control-flow and data-flow relations between activities, using a specialized language such as the Business Process Execution Language (WS-BPEL) or the Business Process Modeling Notation (BPMN).

Definition 1 (Process). A process $\mathcal{P}$ is a tuple ($\mathcal{O}$, $\mathcal{D}$, $\mathcal{E}c$, $\mathcal{E}d$, $\mathcal{S}$) where

- $\mathcal{O}$ is a non empty set of objects which can be partitioned into disjoint sets of activities $\mathcal{A}$, events ($start$, end) and control patterns $\mathcal{C}p$ (choice, parallel, repeat, etc),
- $\mathcal{D}$ is a set of data,
- $\mathcal{E}c$ is a set of control edges where, $\mathcal{E}c \subset \mathcal{O}\times\mathcal{O}$,
- $\mathcal{E}d$ is a a set of data edges where, $\mathcal{E}d \subset \mathcal{A}\times\mathcal{A}\times\mathcal{D}$,
- $\mathcal{S}$ is the set of services invoked by the process.

In this paper, we assume that the processes are **structured** [5]. Recent work has shown that most unstructured process models can be automatically translated into structured ones [6].

Definition 2 (Activity). An activity $a_i \in \mathcal{A}$ is a tuple (in, out, $prop$) where in, out $\subset \mathcal{D}$ are the set of a_i's inputs and outputs respectively, and $prop$ is the set of a_i's properties (e.g. the role of a_i in the process, the service it invokes, its security level, etc).

The partitioning of a process model with respect to a criterion leads to a set of interconnected partitions, each defines the relationship between the objects it includes. Partitions communicate using the interaction patterns (i.e. send, receive, etc) [7]. Next, we refer to the set of activities of the initial process which respond to the same partitioning criterion λ (e.g. activities having the same security level) as $\mathcal{A}_i \mid \cup\mathcal{A}_i = \mathcal{A}$ and $\forall i, j, i \neq j, \mathcal{A}_i \cap \mathcal{A}_j = \{\emptyset\}$.

Definition 3 (Partition). A partition $\mathcal{P}_i$ is a tuple ($\mathcal{O}_i$, $\mathcal{D}_i$, $\mathcal{E}c_i$, $\mathcal{E}d_i$, $\mathcal{S}_i$) where

- $\mathcal{O}_i$ is a set of objects $\mathcal{O}_i \subset \mathcal{O}\cup\mathcal{I}_i$, where $\mathcal{I}_i$ is the set of interaction patterns used by $\mathcal{P}_i$ (e.g. in Fig.2: $send_{d1}$, $receive_{sync2}$, etc).
- $\mathcal{D}_i \subset \mathcal{D} \cup \mathcal{S}ync$, where $\mathcal{S}ync$ is the set of data used for synchronization with other partitions (e.g. in Fig. 2: $sync1$, $sync2$, etc).
- $\mathcal{E}c_i$ is the set of control edges, $\mathcal{E}_c \subset \mathcal{O}_i\times\mathcal{O}_i$
- $\mathcal{E}d_i$ is the set of data edges, $\mathcal{E}_d \subset(\mathcal{A}_i\times\mathcal{A}_i) \cup (\mathcal{I}_i\times(\mathcal{I}\backslash\mathcal{I}_i))$.
- $\mathcal{S}_i \subset \mathcal{S}$ is the set of services invoked by $\mathcal{P}_i$.

In the following, we define a **fragment** as a structured single entry - single exit sub-graph of a process or a partition model.

Definition 4 (partitioning function). The *Process partitioning* is a total function $f_\lambda : \pi \rightarrow \{\pi_i\}^{i=1..\mathcal{N}}$ that takes a centralized process model $\mathcal{P}$ and produces a set of decentralized partitions $\{\mathcal{P}_i\}$ using the decentralization criterion λ. Next, we extend this definition to take into consideration fragments partitioning.

Definition 5 (Preset, postset, transitive preset, transitive postset). We define the $preset$ ($postset$) of an activity a_i, denoted $\bullet a_i$ ($a_i\bullet$), as the set of activities which may execute **just** before (after) a_i and directly linked to it by a set of control dependencies (e.g. in Fig.1 $\bullet a_4 = \{a_1, a_2, a_3\}$). We also define the transitive *preset* (resp., transitive *postset*) of an activity a_i **according to** a partitioning criteria λ and denoted $\bullet T_\lambda\_a_i$ ($T_\lambda\_a_i\bullet$), as the set of activities having the same criterion λ and which may execute just before (after) a_i, and linked to it by a set of control patterns or activities with different criteria. The transitive preset of an activity in the centralized process model represents the preset of the same activity in the corresponding partition (e.g. in Fig.1, $\bullet T_\lambda\_a_4 = \{a_0\}$

Change pattern	Description
Insert(fragment,entry,exit)	Inserts a new fragment into the process between the $entry$ and $exit$ edges in the centralized process model.
Delete(entry,exit)	Deletes a fragment between the $entry$ and $exit$ edges in the centralized process model.
Replace(fragment,entry,exit)	Replaces the existing fragment between $entry$ and $exit$ edges in the centralized process model by a new $fragment$.
Update(activity, prop)	Update an activity's properties. For instance, its security level, its role or the service it invokes.

TABLE I
CHANGE PATTERNS

since a_4 and a_1 belong to the same partition $\mathcal{P}_1$; c.f. Fig. 2). Next, we extend the relations in Definition 5 to that between fragments. For instance the preset of a fragment $\mathcal{F}$ is the smallest fragment including all activities which can be executed just before it and directly connected to it.

IV. Specification and Propagation of Change Operations

In general, process models can be decomposed into SESE fragments [5]. A SESE fragment is a non-empty subgraph in the process model with a single entry and a single exit edge. For every change in the process model, there is at least one enclosing fragment. Here, we consider only the smallest fragment that encloses the changes. This can be achieved using the process structure tree (PST) [5]. In the following, we consider that the fragments enclosing the changes are already identified. In this work, we consider a set of basic change patterns (c.f. Table I), based on which complex change patterns can be expressed [8]. We also assume the well-behavedeness of the updates propagated by the business analysts. It means that the graph production on a business process model are consistent with the behavioral requirements.

In the following, we demonstrate how to propagate the changes made on a centralized specification of a web service orchestration to its resulting decentralized partitions. A change operation on the centralized process model is translated into several change operations each related to a partition. We also consider a process model $\mathcal{P}$ and its derived partitions $\{\mathcal{P}_i\}^{i=1..n}$ according to decentralization function f_λ. An activity is assigned to a partition $\mathcal{P}_i$ only if it responds to criterion λ. Next, we call $\mathbf{g}_\lambda$ the function which maps each activity to a partition. We assume that an activity can not be assigned to more than one partition. It should be noted that the $Replace$ change pattern can be replaced by the two consecutive operations $Delete$ and $Insert$. However, during the change propagation, the number of derived operations resulted from the $Replace$ pattern is less or equal to those derived from the concatenation of the $Delete$ and $Insert$.

A. Change Pattern: Insert$_\mathcal{P}(\mathcal{F}', entry, exit)$

The insertion of new fragment in $\mathcal{P}$ implies the insertion of new activities with different criteria, connected through data and control flows. In this paper, we consider the insertion in sequence but can be easily extended to take into consideration the insertion in parallel or with exclusiveness. The first step is then to identify the partitions affected by the change using g_λ. To achieve this, we look for each partition which respond to the same criterion of at least one activity of $\mathcal{F}$, $\bullet\mathcal{F}$ or $\mathcal{F}\bullet$. Indeed, if we consider two activities a and b in sequence in the centralized process model $\mathcal{P}$, such that they are assigned to different partitions $\mathcal{P}_1$ and $\mathcal{P}_2$ after partitioning. In this case, a and b would communicate through message exchange. If we insert a new activity c between them such it is assigned to a new partition $\mathcal{P}_3$, then we have also to update the old communication between a and b. Therefore $\mathcal{P}_1$ and $\mathcal{P}_2$ are also concerned by the change. Once all affected partitions identified, the second step consists in computing what to insert in each partition as well as the exact position for insertion. The idea is then to partition the new inserted fragment $\mathcal{F}$ according to the same partitioning function f_λ and for each sub-fragment determine the exact position. We consider $\{\mathcal{F}_i\}^{i=1..k}$ the derived sub-fragments where a sub-fragment $\mathcal{F}_i$ should be inserted in partition $\mathcal{P}_i$. Note that the insertion may result in the creation of a new partition. To insert $\mathcal{F}_i$ in $\mathcal{P}_i$, we first compute the transitive preset and postset of $\mathcal{F}_i$ according to λ in $\mathcal{P}$ ($\bullet T_\lambda\_\mathcal{F}_i$, $T_\lambda\_\mathcal{F}_i\bullet$). Note that $\bullet T_\lambda\_\mathcal{F}_i$ and $T_\lambda\_\mathcal{F}_i\bullet$ are **directly** connected in $\mathcal{P}_i$ via control or interaction patterns (e.g. in Figure 2, a_0 and a_4 of $\mathcal{P}_1$ are directly linked by interaction patterns while they were not directly connected in the centralized model). Then, the exact position for the insertion of $\mathcal{F}_i$ in $\mathcal{P}_i$ is between $\bullet T_\lambda\_\mathcal{F}_i$ and $T_\lambda\_\mathcal{F}_i\bullet$. The problem now is how to connect $\mathcal{F}_i$ with $\bullet T_\lambda\_\mathcal{F}_i$ and $T_\lambda\_\mathcal{F}_i\bullet$. Indeed, the latter may already have other fragments or activities between them (e.g. two fragments in parallel with the same properties according to λ have the same transitive $preset$ and $postset$). In this case, we have to identify the relations of $\mathcal{F}_i$ with $\bullet T_\lambda\_\mathcal{F}_i$, $T_\lambda\_\mathcal{F}_i\bullet$ and possibly the fragments between them. For this purpose, we calculate the control paths linking $\mathcal{F}_i$ to $\bullet T_\lambda\_\mathcal{F}_i$ and $T_\lambda\_\mathcal{F}_i\bullet$. Some of the control patterns of these paths may already exist in $\mathcal{P}_i$. To deal with this, we use a union function to merge $\mathcal{F}_i$ with the the fragments that may exist between its transitive $preset$ and $postset$. Finally, an update is required, if necessary, to update the connections between $\bullet T_\lambda\_\mathcal{F}_i$ ($T_\lambda\_\mathcal{F}_i\bullet$) and its $postset$ ($preset$) which may be on other partitions.

B. Change Pattern: Delete$_\mathcal{P}(\mathcal{F}.entry, \mathcal{F}exit)$

The delete removes the set of activities enclosed in the fragment $\mathcal{F} \in\mathcal{P}$. These activities are distributed over the partitions and linked through interaction or control patterns. In the centralized model, the deletion of $\mathcal{F}$ implies the deletion of its links with its $preset$ and $postset$ and the connection of the latter with each other. In the decentralized model, activities of $\mathcal{F}$ are partitioned over partitions and the deletion of an activity a implies the update of its links with $a\bullet$ and $\bullet a$ which may be in different partitions, and possibly with $T_\lambda\_a\bullet$ and $\bullet T_\lambda\_a$ which are in the same partition.

To cope with this, we partition $\mathcal{F}$, identify the position of each $\mathcal{F}_i$ in the respective partition $\mathcal{P}_i$ and delete it. Indeed, since the partitioning function is idempotent, then if $\mathcal{F} \in \mathcal{P}$, $f_\lambda(\mathcal{F})$ is a subgraph of $f_\lambda(\mathcal{P})$. In the centralized process

Change pattern	Change actions
$Insert_{\mathcal{P}}(\mathcal{F}', entry, exit)$	$\forall\ \mathcal{F}_i\prime \in f_\lambda(\mathcal{F}\prime)$, $(entry_i, exit_i) \leftarrow PositionOf(\mathcal{F}_i\prime, \mathcal{P}_i)$ $\forall\ \mathcal{F}_i\prime \in f_\lambda(\mathcal{F}\prime)$, $Insert_{\mathcal{P}_i}(\mathcal{F}_i\prime, entry_i, exit_i)$ $\forall a_j \in \bullet\mathcal{F}\prime$, $update\_connection(a_j, \mathcal{F}\prime)$ $\forall a_k \in \mathcal{F}\prime\bullet$ $update\_connection(\mathcal{F}\prime, a_k)$
$Delete_{\mathcal{P}}(\mathcal{F}.entry, \mathcal{F}exit)$	$\forall\ \mathcal{F}_i \in f_\lambda(\mathcal{F}\prime)$, $(entry_i, exit_i) \leftarrow PositionOf(\mathcal{F}_i, \mathcal{P}_i)$ $\forall\ \mathcal{F}_i \in f_\lambda(\mathcal{F}\prime)$, $Delete_{\mathcal{P}_i}(entry_i, exit_i)$ $\forall a_j \in \bullet\mathcal{F}$, $\forall a_k \in \mathcal{F}\bullet$ $update\_connection(a_j, a_k)$
$Replace_{\mathcal{P}}(\mathcal{F}', \mathcal{F}.entry, \mathcal{F}.exit)$	$\forall\ \mathcal{F}_i \in f_\lambda(\mathcal{F})$, $(entry_i, exit_i) \leftarrow PositionOf(\mathcal{F}_i, \mathcal{P}_i)$ $\forall \mathcal{F}_i \in f_\lambda(\mathcal{F})$,$\mathcal{F}_j\prime \in f_\lambda(\mathcal{F}\prime)$. if $\mathcal{P}_i{=}\mathcal{P}_j$, $Update_{\mathcal{P}_i}(\mathcal{F}_j\prime, entry_i, exit_i)$ $\forall \mathcal{F}_j\prime \in f_\lambda(\mathcal{F}\prime))$ s.t. $\nexists \mathcal{F}_i \in f_\lambda(\mathcal{F})) \wedge p_i \neq p_j$, $Insert_{p_j}(\mathcal{F}_j\prime, entry_j, exit_j)$ where $(entry_j, exit_j) \leftarrow PositionOf(\mathcal{F}_j\prime, p_j)$ $\forall \mathcal{F}_i \in f_\lambda(\mathcal{F}))$ s.t. $\nexists \mathcal{F}_j\prime \in f_\lambda(\mathcal{F}\prime) \wedge p_i \neq p_j$, $Delete_{p_j}(\mathcal{F}_i, entry_i, exit_i)$ where $(entry_i, exit_i) \leftarrow PositionOf(\mathcal{F}_j, p_i)$ $\forall a_j \in \bullet\mathcal{F}$, $update\_connection(a_j, \mathcal{F}\prime)$ $\forall a_k \in \mathcal{F}\bullet$ $update\_connection(\mathcal{F}\prime, a_k)$
$Update_{\mathcal{P}}(a, prop')$	if we consider $a\prime$ as the updated activity then, if $f_\lambda(a) \neq f_\lambda(a')$ then $Replace_{\mathcal{P}}(a', a.entry, a.exit)$

TABLE II
CHANGE PROPAGATION ACTIONS

model, if $\bullet\mathcal{F}_i$ (reps. $\mathcal{F}_i\bullet$) $\notin\mathcal{F}$, then we update the decentralized model linking $\bullet\mathcal{F}_i$ to $\mathcal{F}_i\bullet$ instead of $\mathcal{F}_i$ (reps. $\mathcal{F}_i\bullet$ to $\bullet\mathcal{F}_i$). Note that reduction rules may be applied to the changed partitions to eliminate unnecessary control or interaction patterns.

C. Change Pattern: $Replace_{\mathcal{P}}(\mathcal{F}', \mathcal{F}.entry, \mathcal{F}.exit)$

This pattern replaces an existing fragment $\mathcal{F}$ by a new one $\mathcal{F}\prime$ in the centralized process model. To propagate this change to the concerned partitions, we decentralize $\mathcal{F} = \{\mathcal{F}_i\}^{i=1..l}$ and $\mathcal{F}' = \{\mathcal{F}'_i\}^{i=1..k}$ using f_λ. According to the derived sub-fragments, we figure out two possible scenarios for each fragment of $\mathcal{F}$ and $\mathcal{F}'$; sub-fragments of $\mathcal{F}$ are either deleted or replaced and sub-fragments of $\mathcal{F}\prime$ are either inserted or used to replace existing sub-fragments of $\mathcal{F}$. These scenarios are combined as follows.

- If two sub-fragments $\mathcal{F}_i$ and $\mathcal{F}'_i$ refers to the same partition $\mathcal{P}_i$ then we derive $Replace_{\mathcal{P}_i}(\mathcal{F}\prime_i, \mathcal{F}_i.entry, \mathcal{F}_i.exit)$.
- If a sub-fragment $\mathcal{F}_i$ refers to $\mathcal{P}_i$ such that no $\mathcal{F}'_i$ refers to the same $\mathcal{P}_i$, then we derive $Delete_{\mathcal{P}_i}(\mathcal{F}_{Pi}.entry, \mathcal{F}_i.exit)$.
- If a sub-fragment $\mathcal{F}\prime_i$ refers to $\mathcal{P}_i$ such that no $\mathcal{F}_i$ refers to the same $\mathcal{P}_i$, then we derive $Insert_{\mathcal{P}_i}(\mathcal{F}\prime_i, entry, exit)$. Variables $entry$ and $exit$ are $(\bullet T\_\mathcal{F}\prime_i).exit$ and $(T\_\mathcal{F}\prime_i\bullet).entry$ respectively.

Note that an update phase is required to update the existing links with the modified sub-fragments. This update is similar to the $Insert$ pattern mentioned previously. Formally, the replacement of $\mathcal{F}_i$ by $\mathcal{F}'_i$ corresponds to the deletion of all objects $o \in \mathcal{O}_{\mathcal{F}_i}$, edges $e \in \mathcal{E}c_{\mathcal{F}_i} \cup \mathcal{E}d_{\mathcal{F}_i}$ and data exchanges, and their substitution by the objects, edges and data of $\mathcal{F}\prime_i$. Besides, the connection with its $preset$, $postset$, transitive $preset$ and transitive $postset$ should be updated.

D. Change Pattern: $Update_{\mathcal{P}}(a, prop')$

This pattern updates the properties of one activity (e.g. its security level, its role,etc.). According to a partitioning function f_λ, the properties determine to which partition an activity would be assigned. This leads to two scenarios; (i) The new properties $prop'$ are invariant with respect to the decentralization criterion (e.g. we change the security level while λ is partitioning according to role), or (ii) $prop'$ is variant and then the activity should be moved to another partition. In the latter case, we can either use sequently a $Delete$ then $Insert$ or simply the $Replace$ pattern.

Table II resumes the main and simplified formal actions to achieve the change propagation. In this table, f_λ represents the partitioning function, $PositionOf$ returns the position of a fragment or an activity in a partition, and $update\_connection$ updates the dependencies links between two activities or fragments. For instance, let's consider two activities a and b in sequence in the centralized process model, such they belong to different partitions after decentralization. Then, if we insert a new activity c between them, we have to update the link between a and b, in the decentralized setting, by two new links (a, c) and (c, b).

E. Use case: Insurance Process Example

To have a better understanding of the change propagation approach, we refer to the claim handling process example and we consider the Figure 3. We remind that we want to replace the fragment $\mathcal{F}$ by $\mathcal{F}\prime$. As we already mentioned in Section II, only the partitions $\mathcal{P}_1$ and $\mathcal{P}_3$ are concerned by the change. As can be seen in Figure 3, the partitioning of the fragment $\mathcal{F}\prime$ leads to two fragments $\mathcal{F}\prime_1$ and $\mathcal{F}\prime_2$ connected by inter-partition data and control edges (here we use the partitioning function defined in [2]). The partitioning of the fragment $\mathcal{F}$ leads to $\mathcal{F}_1$ which is the same as $\mathcal{F}$ since it contains one activity. Since $\mathcal{F}\prime_1$ and $\mathcal{F}_1$ concern the same partition $\mathcal{P}_1$ then we derive $Replace(\mathcal{F}\prime_1, \mathcal{F}_1.entry, \mathcal{F}_1.exit)$. Only $\mathcal{F}\prime_2$ concerns the partition $\mathcal{P}_3$ and then we have to insert it in $\mathcal{P}_3$. Hence, we compute the position of the insertion which is between $Receive\_sync3$ and $a_5 : DS$. The generated change operation is then $Insert(\mathcal{F}\prime_2, Receive\_sync3, a_5 : DS)$. However, since $\mathcal{F}\prime_2$ should be in parallel with a_5, then we enclose both of them with an $AND\text{-}split/AND\text{-}join$ patterns. The change propagation is translated as follows:
$Replace_{\mathcal{P}}(\mathcal{F}', \mathcal{F}.entry, \mathcal{F}.exit) \Rightarrow Replace_{\mathcal{P}}(\mathcal{F}'_1, \mathcal{F}_1.entry, \mathcal{F}_1.exit) \wedge Insert_{\mathcal{P}}(\mathcal{F}'_2, Receive\_sync3, a_5 : DS))$.

V. EVALUATION OF THE APPROACH

This section presents the properties of our change propagator. Given a well-behaved structural update on the centralized process model and the derived decentralized sub-processes,

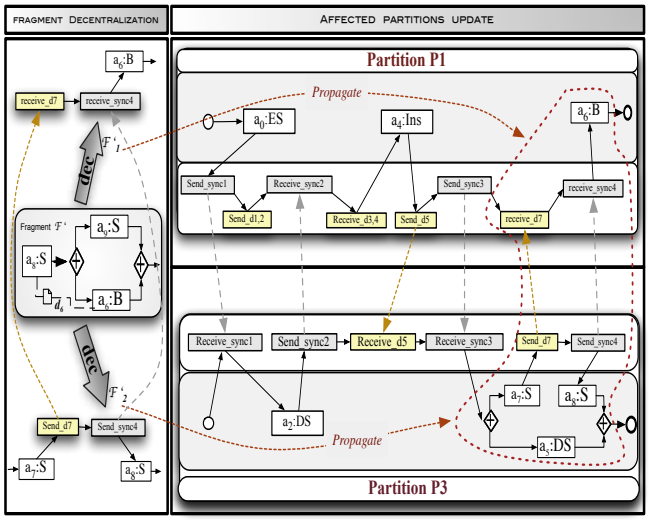

Fig. 3. Change Propagation for the Insurance Example

our approach *automates the change forward propagation* that consistently transforms the update on the source into the related target partitions, as presented in Section IV.

A. Properties and Discussion

As described in Definition 4, the *process partitioning* is a total function of the type $f_\lambda : \pi \rightarrow \{\pi_i\}^{i=1..n}$ that takes a source centralized process $\mathcal{P}$ and produces a target set of decentralized partitions $\{\mathcal{P}_i\}^{i=1..n}$. It establishes a *consistency relation*, denoted $\mathcal{C} \subseteq \pi \times \{\pi_i\}^{i=1..n}$ between the source and the target process models. Since the decentralization algorithm is idempotent, it can be applied multiple times without changing the result, then $\mathcal{C}$ is a total function. $(\mathcal{P}, f_\lambda(\mathcal{P})) \in \mathcal{C}$ means that $\mathcal{P}$ was previously decentralized into $f_\lambda(\mathcal{P})$. Next, we use $\{\mathcal{P}_i\}$ instead of $\{\mathcal{P}_i\}^{i=1..n}$. We use $\Delta_\pi : \pi \rightharpoonup \pi$ and $\Delta_{\pi_\rangle} : \{\pi i\} \rightharpoonup \{\pi_i\}$ as an abbreviation for the update types respectively on the processes and on the partitions. They represent the space of all partial functions describing the changes on each of the centralized and decentralized process models and which can be described by productions, i.e. the change operations defined in Section IV. Now, consider a source change δ that alters $\mathcal{P}$ to $\mathcal{P}'$. The problem is to translate the well-behaved change δ of the source process into a well-behaved changes δ_i on the target partitions, such that the application of both updates results in consistent process models. The change propagator that provides this function is of the type $pr : \pi \times \Delta_\pi \times \{\pi_i\} \rightarrow \{\Delta_{\pi_i}\} \times \{\pi_i\}$. For $\mathcal{P} \in \pi$, $\delta \in \Delta_\pi$ and $f_\lambda(\mathcal{P}) \in \{\pi_i\}$, it computes the changes on the partitions (i.e $\{\delta_i\} \in \{\Delta_{\pi_i}\}$) such that the updated models are consistent (i.e. $(\delta(\mathcal{P}), \{\delta_i(\mathcal{P}_i)\}) \in \mathcal{C}$).

In our semantics, a process and its decentralization result (i.e., the derived partitions) are specified with graphs as introduced in Section II. Then, a change on a process implies a modification on the graph structure which can be expressed by *graph rewriting* rules [9]. Formally, given a graph $\mathcal{G}$, a graph rewriting rule (i.e., also called production) consists of injective morphisms of the form $\delta_\mathcal{G} : \mathcal{L} \rightarrow \mathcal{R}$ that transform a source graph $\mathcal{L}$ into a target graph $\mathcal{R}$. In order to apply this rewrite rule to the initial graph $\mathcal{G}$, a match $m : \mathcal{L} \rightarrow \mathcal{G}$ is needed to specify which part of $\mathcal{G}$ is being updated. Then, the application of $\delta_\mathcal{G}$ to $\mathcal{G}$ via a match m for $\delta_\mathcal{G}$ is uniquely defined by the graph rewriting $\mathcal{G} \Rightarrow_{\delta_\mathcal{G}, m} \mathcal{H}$. This rule application induces a co-match $m' : \mathcal{R} \rightarrow \mathcal{H}$ which specifies the embedding of $\mathcal{R}$ in the result graph $\mathcal{H}$.

The most important criteria is change propagation correctness: a graph-based change propagator must return consistent process models. In this paper, we suppose that when applying a rewriting rule to a given graph $\mathcal{G}$, it is enough to consider the case where the morphisms that matches $\mathcal{L}$ to $\mathcal{G}$ is injective, and that the match m is a total label-preserving, type-preserving and root-preserving [9] graph morphism. However, to be correctly applied, the productions must satisfy the structural consistency of the centralized process constraints. Note that we assume the well-behavedeness of the updates propagated by the designers. It means that the graph production on a centralized process is consistent with the behavioral requirements, and after the production the process remains structured. Moreover, the fragment or process partitioning preserves by definition the well-behaved process semantic. Secondly, a fundamental law is that the change propagation should be deterministic: for each centralized process model input there is a unique decentralization result. In our case, the change propagator is modeled by a mathematical function. Given the same pair of the centralized and its decentralized models, and a finite set of changes (i.e., bounded within the SESE fragment) on the source centralized process model, our propagator produces the same changes on the target partitions.

Finally, to adapt $f_\lambda(\mathcal{P})$ to the changes induced by δ without re-decentralizing afresh the entire updated centralized process model, i.e. $f_\lambda(\delta(\mathcal{P}))$, our change propagator enforces an in-place synchronization between $\delta_(\mathcal{P})$ and $f_\lambda(\mathcal{P})$ by translating the updates δ into well-behaved target updates (δ_i) to get $(\delta_i(\mathcal{P}_i)))$ consistent with $\delta(\mathcal{P})$. The *change translation* is a partial function of the type $\Delta_\pi \rightharpoonup \{\Delta_{\pi_i}\}$. Indeed, the propagation of δ does not affect all partitions which make the complexity of our approach lower than the re-decentralization of the whole process coupled with *diff*-based methods [9].

B. Proof of Concepts Prototype

The *change propagator* has been implemented and integrated with our previous development of the *partitioning algorithm* [2] as an extension to a BPMN Editor [10]. This BPMN editor is based on a graph visualization library, and it is used to model a source centralized process model, for instance the structured process of Figure 1. After applying our *partitioning algorithm*, we obtain the partitions depicted in Figure 2 using the graph library. Moreover, we have developed a filter that logs the process model editing operations presented in Section IV. Actually, the specification of the entry and the exit of a fragment is performed manually, for example as depicted in Figure 3, but it can be easily automated. The *change propagation algorithm* is implemented in the *DROOLS* [11] inference engine, and it automatically computes the graph editing operation sequence that manipulates the partitions.

VI. Related Work

The topic of this paper is change in business processes. There is a multitude of approaches dealing with related issues, ranging from ad-hoc changes of single process instances to evolutionary changes of the entire process description [12].

In a **centralized process setting**, all design and runtime information are available (e.g., process model and state of running process instances) and the process orchestration is not fragmented. In this basic setting, major challenges are correctness of the applied changes, efficient migration of running process instances to modified process descriptions, as well as proper inclusion of users [13], [12]. Nowadays, there are even fully adaptive process management systems available, e.g., AristaFlow [14]. The main difference of the approach presented in this paper is obviously the fragmentation of the process orchestration, hence imposing new questions when compared to the central setting. However, many things can be transferred from the central case such as the need for correctness considerations and the implementation of change patterns as proposed in [8] for decentralized process settings.

In the **decentralized setting**, [5] presents a formal model for a distributed workflow change management (DWFCM) that uses a rules topic ontology and a service ontology to support the needed run-time flexibility. The approach aims to generate a new workflow that is migration consistent with the original workflow. This work is different from our proposal, since they do not seek to propagate a pre-defined changes on a centralized process to that on the derived partitions. In [15] the authors present a unidirectional model incremental transformation approach. Its central contribution is the definition and the realization of an automatic synchronizer for managing and re-establishing the structural consistency of heterogeneous source and target models. Other approaches addressing flexibility and change in decentralized or – as referred to in these papers – distributed process settings focus on the ad-hoc modification of single process instances at runtime [16], [17]. The difference to our work presented is that these approaches do not physically change the partitions but just migrate some instances.

In **WS choreographies** there are few approaches addressing change and evolution . In DYCHOR [18], for example, it is investigated how a change initiated at one partner's side can be propagated to the other partners of the choreography. One similar approach is presented in [19] where a structured model using RPST is used to model choreographies as well as the public and private views. The authors investigated structural and semantical propagation and dialed with the transitive effects of the changes. At a general level, techniques for evolving partitioned process settings exploit the knowledge on the different partitions such that they cannot be applied to choreographies where this knowledge is not at hand. Vice versa, techniques for evolving choreographies (e.g., [18], [19]) could be applied to partitioned process settings, however this would require the construction of artificial private and public views as well as choreography model resulting in an unnecessary overhead.

VII. Conclusion

In this paper, we presented an approach to propagate changes form a centralized process to its derived decentralized partitions. The proposed approach is based on four basic change patterns and computes the partitions involved in the change as well as the regions to be modified. It also translates the initial change operation on the centralized process into several change operations for the partitions affected by the change. The introduced change operations can be composed to give rise to more complex change patterns with enhanced semantics (e.g., move of fragment, refactoring of fragments: splitting and merging). In worst case, the propagation of changes is equal in complexity to the re-decentralization of the whole process (i.e. the smallest fragment that encloses the change is equal to the process model). As a future work, we plan to study the impacts and management of many running versions of the partitions affected by the change.

References

[1] R. Khalaf, "Supporting business process fragmentation while maintaining operational semantics: a BPEL perspective," Ph.D. dissertation, University of Stuttgart, 2008.

[2] W. Fdhila, U. Yildiz, and C. Godart, "A flexible approach for automatic process decentralization using dependency tables," in *ICWS*, 2009, pp. 847–855.

[3] W. Fdhila, A. Baouab, K. Dahman, C. Godart, O. Perrin, and F. Charoy, "Change propagation in decentralized composite web services," in *CollaborateCom*, 2011, pp. 508–511.

[4] W. Fdhila, M. Dumas, and C. Godart, "Optimized decentralization of composite web services," in *CollaborateCom*, 11-14 2010, pp. 1 –10.

[5] J. Vanhatalo, H. Völzer, and F. Leymann, "Faster and more focused control-flow analysis for business process models through sese decomposition," in *ICSOC*, 2007, pp. 43–55.

[6] A. Polyvyanyy, L. Garcia-Banuelos, and M. Dumas, "Structuring acyclic process models," *Information Systems*, vol. 37, no. 6, pp. 518–538, 2012.

[7] A. P. Barros, M. Dumas, and A. H. M. ter Hofstede, "Service interaction patterns," in *Business Process Management*, 2005, pp. 302–318.

[8] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features - enhancing flexibility in process-aware information systems," *Data Knowl. Eng.*, vol. 66, no. 3, pp. 438–466, 2008.

[9] E. Biermann, C. Ermel, and G. Taentzer, "Precise Semantics of EMF Model Transformations by Graph Transformation," in *MoDELS*, 2008, pp. 53–67.

[10] Yaoqiang, "Open source bpmn 2.0 modeler: bpmn.yaoqiang.org," 2011.

[11] Drools, "labs.jboss.com/drools/," (Feb. 2011).

[12] S. Rinderle, M. Reichert, and P. Dadam, "Correctness criteria for dynamic changes in workflow systems - a survey," *Data Knowl. Eng.*, vol. 50, no. 1, pp. 9–34, 2004.

[13] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. Aalst, "Process flexibility: A survey of contemporary approaches," in *Advances in Enterprise Engineering I*, 2008, pp. 16–30.

[14] P. Dadam and M. Reichert, "The ADEPT project: A decade of research and development for robust and flexible process support - challenges and achievements," *Computer Science - Research and Development*, vol. 23, no. 2, pp. 81–97, 2009.

[15] K. Dahman, F. Charoy, and C. Godart, "Towards consistency management for a business-driven development of soa," in *EDOC*, 2011.

[16] M. Reichert and T. Bauer, "Supporting ad-hoc changes in distributed workflow management systems," in *CoopIS*, 2007, pp. 150–168.

[17] V. Atluri and S. Chun, "Handling dynamic changes in decentralized workflow execution environments," in *Database and Expert Systems Applications*, 2003, pp. 813–825.

[18] S. Rinderle, A. Wombacher, and M. Reichert, "Evolution of process choreographies in DYCHOR," in *CoopIS*, 2006, LNCS, pp. 273–290.

[19] W. Fdhila, S. Rinderle-Ma, and M. Reichert, "Change propagation in collaborative processes scenarios," in *CollaborateCom*, 2012.